24 | 文本分类：如何使用BERT构建文本分类模型？

2021-12-06 方远

《PyTorch深度学习实战》 课程介绍 >

讲述：方远

时长 15:39 大小 14.35M

你好，我是方远。

在第 22 节课我们一起学习了不少文本处理方面的理论，其实文本分类在机器学习领域的应用也非常广泛。

比如说你现在是一个 NLP 研发工程师，老板啪地一下甩给你一大堆新闻文本数据，它们可能来源于不同的领域，比如体育、政治、经济、社会等类型。这时我们就需要对文本分类处理，方便用户快速查询自己感兴趣的内容，甚至按用户的需要定向推荐某类内容。

这样的需求就非常适合用 PyTorch + BERT 处理。为什么会选择 BERT 呢？因为 BERT 是比较典型的深度学习 NLP 算法模型，也是业界使用最广泛的模型之一。接下来，我们就一起来搭建这个文本分类模型，相信我，它的效果表现非常强悍。

问题背景与分析

正式动手之前，我们不妨回顾一下历史。文本分类问题有很多经典解决办法。

开始时就是最简单粗暴的关键词统计方法。之后又有了基于贝叶斯概率的分类方法，通过某些条件发生的概率推断某个类别的概率大小，并作为最终分类的决策依据。尽管这个思想很简单，但是意义重大，时至今日，贝叶斯方法仍旧是非常多应用场景下的好选择。

之后还有支持向量机（SVM），很长一段时间，其变体和应用都在 NLP 算法应用的问题场景下占据统治地位。

随着计算设备性能的提升、新的算法理论的产生等进步，一大批的诸如随机森林、LDA 主题模型、神经网络等方法纷纷涌现，可谓百家争鸣。

既然有这么多方法，为什么这里我们这里推荐选用 BERT 呢？

因为在很多情况下，尤其是一些复杂场景下的文本，像 BERT 这样具有强大处理能力的工具才能应对。比如说新闻文本就不好分类，因为它存在后面这些问题。

1.**类别多**。在新闻资讯 App 中，新闻的种类是非常多的，需要产品经理按照统计、实用的原则进行文章分类体系的设计，使其类别能够覆盖所有的文本，一般来说都有 50 种甚至以上。不过为了让你把握重点，咱们先简化问题，假定文本的分类体系已经确定。

2.**数据不平衡**。不难理解，在新闻中，社会、经济、体育、娱乐等类别的文章数量相对来说是比较多的，占据了很大的比例；而少儿、医疗等类别则相对较少，有的时候一天也没有几篇对应的文章。

3.**多语言**。一般来说，咱们主要的语言除了中文，应该是大多数人只会英语了，不过为了考虑到新闻来源的广泛性，咱们也假定这批文本是多语言的。

刚才提到了，因为 Bert 是比较典型的深度学习 NLP 算法模型，也是业界使用最广泛的模型之一。如果拿下这么有代表性的模型，以后你学习和使用基于 Attention 的模型你也能举一反三，比如 GPT 等。

想要用好 BERT，我们需要先了解它有哪些特点。

BERT 原理与特点分析

BERT 的全称是 Bidirectional Encoder Representation from Transformers，即双向 Transformer 的 Encoder。作为一种基于 Attention 方法的模型，它最开始出现的时候可以说是抢尽了风头，在文本分类、自动对话、语义理解等十几项 NLP 任务上拿到了历史最好成绩。

在第 22 节课（如果不熟悉可以回看），我们已经了解了 Attention 的基本原理，有了这个知识做基础，我们很容易就能快速掌握 BERT 的原理。

这里我再快速给你回顾一下，BERT 的理论框架主要是基于论文《Attention is all you need》中提出的 Transformer，而后者的原理则是刚才提到的 Attention。**其最为明显的特点，就是摒弃了传统的 RNN 和 CNN 逻辑，有效解决了 NLP 中的长期依赖问题。**

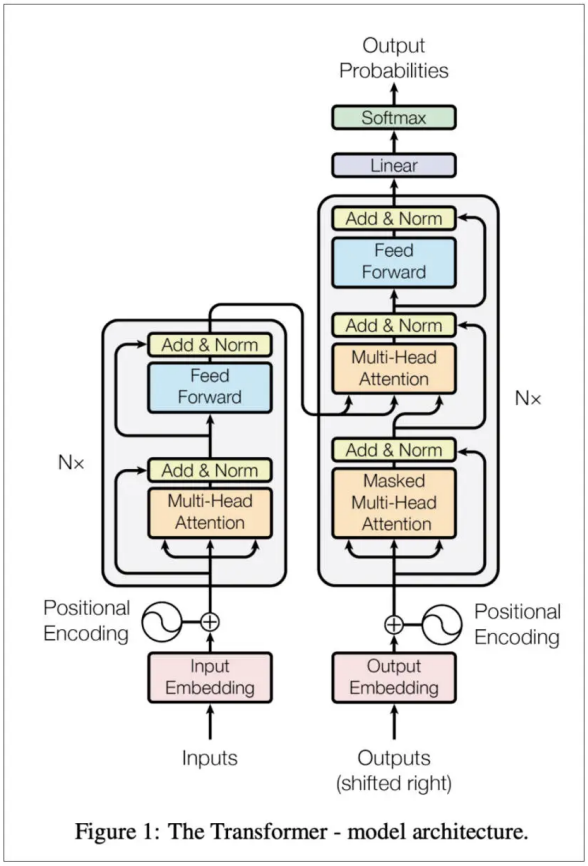


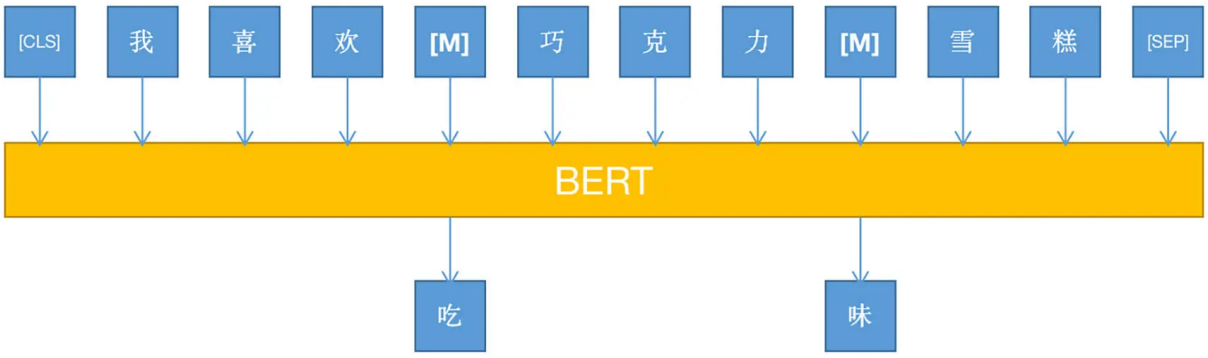
Figure 1: The Transformer - model architecture.

图片来源：https://arxiv.org/abs/1706.03762

在 BERT 中，它的输入部分，也就是图片的左边，其实是由 N 个多头 Attention 组合而成。多头 Attention 是将模型分为多个头，形成多个子空间，可以让模型去关注不同方面的信息，这有助于网络捕捉到更丰富的特征或者信息。（具体原理，一定要查阅《Attention is all you need》哦）。

结合上图我们要注意的是，BERT 采用了基于 MLM 的模型训练方式，即 Mask Language Model。因为 BERT 是 Transformer 的一部分，即 encoder 环节，所以没有 decoder 的部分（其实就是 GPT）。

为了解决这个问题，MLM 方式应运而生。它的思想也非常简单，就是在**训练之前，随机将文本中一部分的词语（token）进行屏蔽（mask），然后在训练的过程中，使用其他没有被屏蔽的 token 对被屏蔽的 token 进行预测。**



用过 Word2Vec 的小伙伴应该比较清楚，在 Word2Vec 中，对于同一个词语，它的向量表示是固定的，这也就是为什么会有那个经典的“*国王 - 男人 + 女人 = 皇后*”计算式了。

但是有一个问题，“苹果”这个词，有可能是水果的苹果，也可能是电子产品的品牌，如果还是用同一个向量表示，这样就有可能产生偏差。而在 BERT 中则不一样，根据上下文的不同，对于同一个 token 给出的词向量是动态变化的，更加灵活。

此外，BERT 还有多语言的优势。在以前的算法中，比如 SVM，如果要做多语言的模型，就要涉及分词、提取关键词等操作，而这些操作要求你对该语言有所了解。像阿拉伯文、日语等语言，咱们大概率是看不懂的，这会对我们最后的模型效果产生极大影响。

BERT 则不需要担心这个问题，通过基于字符、字符片段、单词等不同粒度的 token 覆盖并作 WordPiece，能够覆盖上百种语言，甚至可以说，只要你能够发明出一种逻辑上自洽的语言，BERT 就能够处理。有关 WordPiece 的介绍，你可以通过这里做拓展阅读。

好，说了这么多，集高效、准确、灵活再加上用途广泛于一体的 BERT，自然而然就成为了咱们的首选，下面咱们开始正式构建一个文本分类模型。

安装与准备

工欲善其事，必先利其器，在开始构建模型之前，我们要安装相应的工具，然后下载对应的预先训练好的模型，同时还要了解数据的格式。

环境准备

因为咱们要做的是一个基于 PyTorch 的 BERT 模型，那么就要安装对应的 python 包，这里我选择的是 hugging face 的 PyTorch 版本的 Transformers 包。你可以通过 pip 命令直接安装。

```
pip install Transformers
```

复制代码

模型准备

安装之后，我们打开 Transformers 的git 页面，并找到如下的文件夹。

```
src/Transformers/models/BERT
```

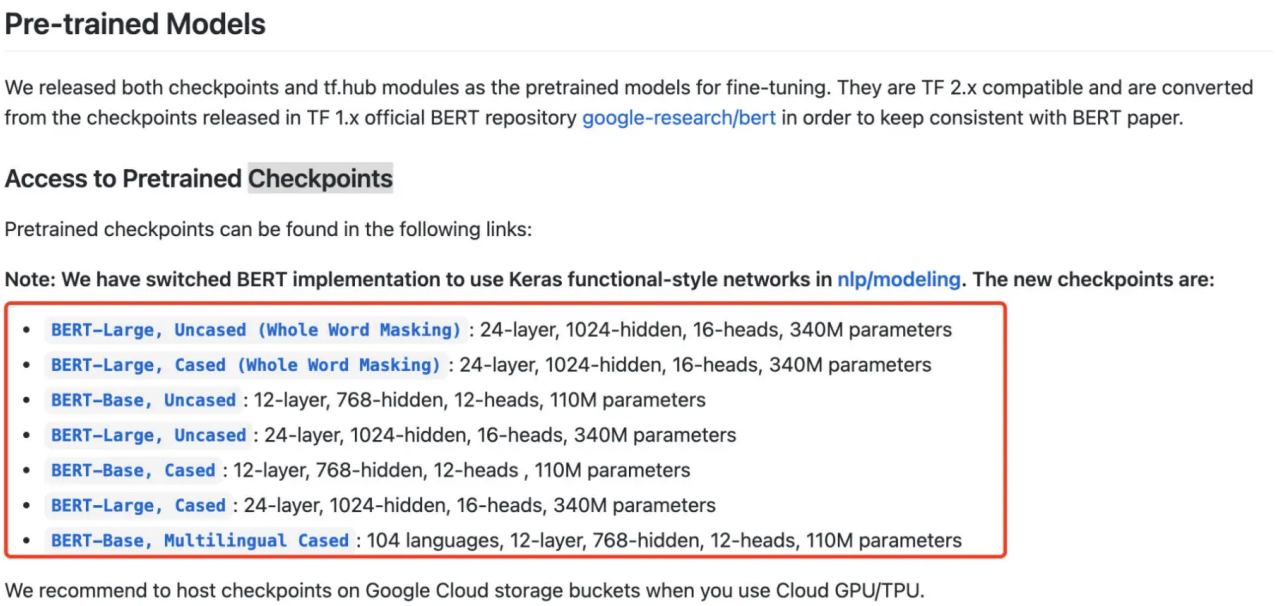
复制代码

从这个文件夹里，我们需要找到两个很重要的文件，分别是 convert_BERT_original_tf2_checkpoint_to_PyTorch.py 和 modeling_BERT.py 文件。

先来看第一个文件，你看看名字，是不是就能猜出来，它大概是用来做什么的了？没错，就是用来将原来通过 TensorfFlow 预训练的模型转换为 PyTorch 的模型。

然后是 modeling_BERT.py 文件，这个文件实际上是给了你一个使用 BERT 的范例。

下面，咱们开始准备模型，打开这个地址，你会发现在这个页面中，有几个预训练好的模型。



Pre-trained Models

We released both checkpoints and tf.hub modules as the pretrained models for fine-tuning. They are TF 2.x compatible and are converted from the checkpoints released in TF 1.x official BERT repository google-research/bert in order to keep consistent with BERT paper.

Access to Pretrained Checkpoints

Pretrained checkpoints can be found in the following links:

Note: We have switched BERT implementation to use Keras functional-style networks in nlp/modeling. The new checkpoints are:

- `BERT-Large, Uncased (Whole Word Masking)` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Large, Cased (Whole Word Masking)` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Uncased` : 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Large, Uncased` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Cased` : 12-layer, 768-hidden, 12-heads , 110M parameters
- `BERT-Large, Cased` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Multilingual Cased` : 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters

We recommend to host checkpoints on Google Cloud storage buckets when you use Cloud GPU/TPU.

对照这节课的任务，我们选择的是“BERT-Base, Multilingual Cased”的版本。从 GitHub 的介绍可以看出，这个版本的 checkpoint 支持 104 种语言，是不是很厉害？当然，如果你没有多语言的需求，也可以选择其他版本的，它们的区别主要是网络的体积不同。

转换完模型之后，你会发现你的本地多了三个文件，分别是 config.json、pytorch_model.bin 和 vocab.txt。我来分别给你说一说。

1.config.json：顾名思义，该文件就是 BERT 模型的配置文件，里面记录了所有用于训练的参数设置。

2.PyTorch_model.bin：模型文件本身。

3.vocab.txt：词表文件。尽管 BERT 可以处理一百多种语言，但是它仍旧需要词表文件用于识别所支持语言的字符、字符串或者单词。

格式准备

现在模型准备好了，我们还要看看跟模型匹配的格式。BERT 的输入不算复杂，但是也需要了解其形式。在训练的时候，我们输入的数据不能是直接把词塞到模型里，而是要转化成后面这三种向量。

1.**Token embeddings**：词向量。这里需要注意的是，Token embeddings 的第一个开头的 token 一定得是“[CLS]”。[CLS]作为整篇文本的语义表示，用于文本分类等任务。

2.**Segment embeddings**。这个向量主要是用来将两句话进行区分，比如问答任务，会有问句和答句同时输入，这就需要一个能够区分两句话的操作。不过在咱们此次的分类任务中，只有一个句子。

3.**Position embeddings**。记录了单词的位置信息。

模型构建

准备工作已经一切就绪，我们这就来搭建一个基于 BERT 的文本分类网络模型。这包括了**网络的设计、配置、以及数据准备，这个过程也是咱们的核心过程**。

网络设计

从上面提到的 modeling_BERT.py 文件中，我们可以看到，作者实际上已经给我们提供了很多种类的 NLP 任务的示例代码，咱们找到其中的“BERTForSequenceClassification”，这个分类网络我们可以直接使用，它也是最最基础的 BERT 文本分类的流程。

这个过程包括了利用 **BERT 得到文本的 embedding 表示、将 embedding 放入全连接层得到分类结果**两部分。我们具体看一下代码。

复制代码

```
class BERTForSequenceClassification(BERTPreTrainedModel):
    def __init__(self, config):
        super().__init__(config)
        self.num_labels = config.num_labels//类别标签数量
        self.bert = BertModel(config)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)//还记得Dropout是用
        self.classifier = nn.Linear(config.hidden_size, config.num_labels)//BE
        self.init_weights()

    def forward(
```

```
        self,
        input_ids=None,
        attention_mask=None,
        token_type_ids=None,
        position_ids=None,
        head_mask=None,
        inputs_embeds=None,
        labels=None,
        output_attentions=None,
        output_hidden_states=None,
        return_dict=None,
    ):
        outputs = self.bert(
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
            head_mask=head_mask,
            inputs_embeds=inputs_embeds,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
        )

        pooled_output = outputs[1]//这个就是经过BERT得到的中间输出。

        pooled_output = self.dropout(pooled_output)//对，就是为了减少过拟合和增加网
        logits = self.classifier(pooled_output)//多层MLP输出最后的分类结果。

```

对照前面的代码，可以发现，接收到输入信息之后，BERT 返回了一个 outputs，outputs 包括了模型计算之后的全部结果，不仅有每个 token 的信息，也有整个文本的信息，这个输出具体包括以下信息。

last_hidden_state 是模型最后一层输出的隐藏层状态序列。shape 是 (batch_size, sequence_length, hidden_size)。其中 hidden_size=768，这个部分的状态，就相当于利用 sequence_length * 768 维度的矩阵，记录了整个文本的计算之后的每一个 token 的结果信息。

pooled_output，代表序列的第一个 token 的最后一个隐藏层的状态。shape 是 (batch_size, hidden_size)。所谓的第一个 token，就是咱们刚才提到的[CLS]标签。

除了上面两个信息，还有 hidden_states、attentions、cross attentions。有兴趣的小伙伴可以去查一下，它们有何用途。

通常的任务中，我们用得比较多的是 last_hidden_state 对应的信息，我们可以用 pooled_output = outputs[1]来进行获取。

至此，我们已经有了经过 BERT 计算的文本向量表示，然后我们将其输入到一个 linear 层中进行分类，就可以得到最后的分类结果了。**为了提高模型的表现，我们往往会在 linear 层之前，加入一个 dropout 层，这样可以减少网络的过拟合的可能性，同时增强神经元的独立性**。

模型配置

设计好网络，我们还要对模型进行配置。还记得刚才提到的 config.json 文件么？这里面就记录了 BERT 模型所需的所有配置信息，我们需要对其中的几个内容进行调整，这样模型就能知道我们到底是要做什么事情了。

后面这几个字段我专门说一下。

id2label：这个字段记录了类别标签和类别名称的映射关系。

label2id：这个字段记录了类别名称和类别标签的映射关系。

num_labels_cate：类别的数量。

数据准备

模型网络设计好了，配置文件也搞定了，下面我们就要开始数据准备这一步了。这里的数据准备是指将文本转换为 BERT 能够识别的形式，即前面提到的三种向量，在代码中，对应的就是 input_ids、token_type_ids、attention_mask。

为了生成这些数据，我们需要在 git 中找到“src/Transformers/data/processors/utils.py”文件，在这个文件中，我们要用到以下几个内容。

1.InputExample：它用于记录单个训练数据的文本内容的结构。

2.DataProcessor：通过这个类中的函数，我们可以将训练数据集的文本，表示为多个 InputExample 组成的数据集合。

3.get_features：用于把 InputExample 数据转换成 BERT 能够理解的数据结构的关键函数。我们具体来看一下各个数据都怎么生成的。

input_ids 记录了输入 token 对应在 vocab.txt 的 id 序号，它是通过如下的代码得到的。

```
input_ids = tokenizer.encode(
                    example.text_a,
                    add_special_tokens=True,
                    max_length=min(max_length, tokenizer.max_len),
                )
```

而 attention_mask 记录了属于第一个句子的 token 信息，通过如下代码得到。

```
attention_mask = [1 if mask_padding_with_zero else 0] * len(input_ids)
attention_mask = attention_mask + ([0 if mask_padding_with_zero else 1] * padd
```

另外，不要忘记记录文本类别的信息（label）。你可以自己想想看，能否按照 utils.py 文件中的声明方式，构建出对应的 label 信息呢？

模型训练

到目前为止，我们有了网络结构定义（BERTForSequenceClassification）、数据集合（get_features），现在就可以开始编写实现训练过程的代码了。

选择优化器

首先我们来选择优化器，代码如下。我们要对网络中的所有权重参数进行设置，这样优化器就可以知道哪些参数是要进行优化的。然后我们将参数 list 放到优化器中，BERT 使用的是 AdamW 优化器。

```
param_optimizer = list(model.named_parameters())
no_decay = ['bias', 'LayerNorm.bias', 'LayerNorm.weight']
optimizer_grouped_parameters = [
            {'params': [p for n, p in param_optimizer if not any(nd in n for n
            {'params': [p for n, p in param_optimizer if any(nd in n for nd in
        ]
optimizer = AdamW(optimizer_grouped_parameters, lr=args.learning_rate, eps=arg
```

这部分的代码，主要是为了选择一个合适咱们模型任务的优化器，并将网络中的参数设定好学习率。

构建训练过程逻辑

训练的过程逻辑是非常简单的，只需要两个 for 循环，分别代表 epoch 和 batch，然后在最内部增加**一个训练核心语句，以及一个梯度更新语句**，这就足够了。可以看到，PyTorch 在工程代码的实现上，封装得非常完善和简练。

复制代码

```
for epoch in trange(0, args.num_train_epochs):
  model.train()//一定别忘了要把模型设置为训练状态。
  for step, batch in enumerate(tqdm(train_dataLoader, desc='Iteration')):
    step_loss = training_step(batch)//训练的核心环节
    tr_loss += step_loss[0]
    optimizer.step()
    optimizer.zero_grad()
```

训练的核心环节

训练的核心环节，你需要关注两个部分，分别是**通过网络得到预测输出**，也就是 logits，以及**基于 logits 计算得到的 loss**，loss 是整个模型使用梯度更新需要用到的数据。

复制代码

```
def training_step(batch):
  input_ids, token_type_ids, attention_mask, labels = batch
  input_ids = input_ids.to(device)//将数据发送到GPU
  token_type_ids = token_type_ids.to(device)
  attention_mask = attention_mask.to(device)
  labels = labels_voc.to(device)

  logits = model(input_ids,
        token_type_ids=token_type_ids,
        attention_mask=attention_mask,
```

```
        labels=labels)
    loss_fct = BCEWithLogitsLoss()
    loss = loss_fct(logits.view(-1, num_labels_cate), labels.view(-1, num_labels
    loss.backward()
```

至此，咱们已经快速构建出了一个 BERT 分类器所需的所有关键代码。但是仍旧有一些小小的环节需要你来完善，比如 training_step 代码块中的 device，是怎么得到的呢？回顾一下咱们之前学习的内容，相信你一定可以做得到。

小结

恭喜你完成了这节课的学习，尽管现在 GitHub 上已经有了很多已经封装得非常完善的 BERT 代码，你也可以很快实现一个最基本的 NLP 算法流程，但是我仍希望你能够抽出时间，好好看一下 Transformer 中的模型代码，这会对你的技术提升有非常大的助益。

这节课我们学习了如何用 PyTorch 快速构建一个基本的文本分类模型，想要实现这个过程，你需要了解 BERT 的预训练模型的获取以及转化、分类网络的设计方法、训练过程的编写。整个过程不难，但是却可以让你快速上手，了解 PyTorch 在 NLP 方面如何应用。

除了技术本身，业务方面的考虑我们也要注意。比如新闻文本的多语言、数据不平衡等问题，模型有时不能解决所有的问题，因此你还需要学习一些**数据预处理的技巧**，这包括很多技术和算法方面的内容。

即使我列出一份长长的学习清单，也可能会挂一漏万，所以数据预处理方面的知识我建议你重点关注以下内容：建议你需要花一些时间去学习 NumPy 和 Pandas 的使用，这样才能更加得心应手地处理数据；你还可以多学习一些常见的数据挖掘算法（比如决策树、KNN、支持向量机等）；另外，深度学习的广泛使用，其实仍旧非常需要传统机器学习算法的背后支撑，也建议你多多了解。

思考题

BERT 处理文本是有最大长度要求的（512），那么遇到长文本，该怎么办呢？

也欢迎你在留言区记录你的疑问或者收获，也推荐你把这节课分享给你的朋友。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 2 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 23 | 情感分析：如何使用LSTM进行情感分析？

下一篇 25 | 摘要：如何快速实现自动文摘生成？

更多学习推荐

精选留言 (1)

写留言

vcjmhg
2021-12-09

1.截断法
截断法是非常常用办法，大致分为三种，head截断，tail截断，head+tail 截断。
head截断即从文本开头直到限制的字数。
tail截断是从结尾开始往前截断。
head+tail 截断，开头和结尾各保留一部分，比例参数是一个可以调节超参数。...
展开

作者回复: 👍👍👍👍^^